НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет систем управления и робототехники

Операционные системы

Лабораторная работа № 1

Выполнил студент

Ефаринов Павел Андреевич

Группа № P33122

Преподаватели: Клименков Сергей Викторович

Покид Александр Владимирович

г. Санкт-Петербург

2020

**Вариант:** A=186;B=0xA6EF4BD5;C=mmap;D=32;E=112;F=block;G=37;H=seq;I=27;J=sum;K=sem

**Задание:**

Разработать программу на языке С, которая осуществляет следующие действия

* Создает область памяти размером A мегабайт, начинающихся с адреса B (если возможно) при помощи C=(malloc, mmap) заполненную случайными числами /dev/urandom в D потоков. Используя системные средства мониторинга, определите адрес начала в адресном пространстве процесса и характеристики выделенных участков памяти. Замеры виртуальной/физической памяти необходимо снять:

1. До аллокации
2. После аллокации
3. После заполнения участка данными
4. После деаллокации

* Записывает область памяти в файлы одинакового размера E мегабайт с использованием F=(блочного, некешируемого) обращения к диску. Размер блока ввода-вывода G байт. Преподаватель выдает в качестве задания последовательность записи/чтения блоков H=(последовательный, заданный  или случайный)
* Генерацию данных и запись осуществлять в бесконечном цикле.
* В отдельных I потоках осуществлять чтение данных из файлов и подсчитывать агрегированные характеристики данных - J=(сумму, среднее значение, максимальное, минимальное значение).
* Чтение и запись данных в/из файла должна быть защищена примитивами синхронизации K=(futex, cv, sem, flock).
* По заданию преподавателя изменить приоритеты потоков и описать изменения в характеристиках программы.

Для запуска программы возможно использовать операционную систему Windows 10 или  Debian/Ubuntu в виртуальном окружении.

Измерить значения затраченного процессорного времени на выполнение программы и на операции ввода-вывода используя системные утилиты.

Отследить трассу системных вызовов.

Используя stap построить графики системных характеристик.

**Отчет:**

#include <sys/mman.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <string.h>

#include <unistd.h>

const size\_t A = 186 \* 1024 \* 1024;

void\* B = (void\*) 0xA6EF4BD5;

void\* page\_aligned\_B = (void\*) 0xA6EF4000;

const int D = 32;

const int E = 112 \* 1024 \* 1024;

const size\_t G = 37;

const int I = 27;

const int file\_number = 20;

\_Atomic unsigned long long processed\_files\_count = 0;

sem\_t files\_semaphore;

struct memory\_filler\_data {

size\_t part\_length;

void\* address;

FILE\* random\_file;

};

void\* memory\_filler\_thread(void\* in\_thread\_data) {

struct memory\_filler\_data thread\_data = \*(struct memory\_filler\_data\*) in\_thread\_data;

fread(thread\_data.address, thread\_data.part\_length, 1, thread\_data.random\_file);

free(in\_thread\_data);

pthread\_exit(NULL);

}

\_Noreturn void\* file\_aggregator\_thread() {

while (1) {

sem\_wait(&files\_semaphore);

printf("Aggregator thread awakened!\n");

unsigned long long current\_file = processed\_files\_count++;

char filename[20];

sprintf(filename, "%llu", current\_file);

long long sum = 0;

int num;

FILE\* output\_file = fopen(filename, "r");

while (fread(&num, sizeof(int), 1, output\_file) == 1) {

sum += num;

}

fclose(output\_file);

printf("My current file is: %llu It's sum is: %lld\n", current\_file, sum);

remove(filename);

}

pthread\_exit(NULL);

}

void fill\_memory(void\* memory\_address, size\_t memory\_size) {

//before allocation

mmap(memory\_address, memory\_size, PROT\_READ | PROT\_WRITE, MAP\_ANONYMOUS | MAP\_PRIVATE, -1, 0);

//after allocation

FILE\* random\_file = fopen("/dev/urandom", "r");

pthread\_t filler\_threads[D];

size\_t part\_length = memory\_size / D;

for (int i = 0; i < D; ++i) {

struct memory\_filler\_data\* thread\_data = malloc(sizeof(struct memory\_filler\_data));

thread\_data->part\_length = part\_length;

thread\_data->address = memory\_address + part\_length \* i;

thread\_data->random\_file = random\_file;

pthread\_create(&filler\_threads[i], NULL, memory\_filler\_thread, (void\*) thread\_data);

}

for (int i = 0; i < D; ++i) {

pthread\_join(filler\_threads[i], NULL);

}

fclose(random\_file);

//after memory fill

}

void fill\_file\_from\_memory(void\* memory\_address, size\_t data\_length, char\* filename) {

FILE\* output\_file = fopen(filename, "w");

fwrite(memory\_address, G, data\_length / G + 1, output\_file);

fclose(output\_file);

}

void\* fill\_memory\_write\_file() {

sem\_init(&files\_semaphore, 0, 0);

unsigned long long created\_files\_count = 0;

while (1) {

// for (int i = 0; i < file\_number; ++i) {

fill\_memory(B, A);

char\* filename = malloc(20 \* sizeof(char));

sprintf(filename, "%llu", created\_files\_count);

printf("Current file: %s\n", filename);

fill\_file\_from\_memory(B, E, filename);

created\_files\_count++;

sem\_post(&files\_semaphore);

munmap(page\_aligned\_B, A);

free(filename);

// after deallocating

}

sem\_destroy(&files\_semaphore);

pthread\_exit(NULL);

}

void\* aggregate\_files() {

sleep(25);

pthread\_t aggregation\_threads[I];

for (int i = 0; i < D; ++i) {

pthread\_create(&aggregation\_threads[i], NULL, file\_aggregator\_thread, NULL);

}

pthread\_exit(NULL);

}

int main() {

pthread\_t\* memory = malloc(sizeof(pthread\_t));

pthread\_t\* files = malloc(sizeof(pthread\_t));

pthread\_create(memory, NULL, fill\_memory\_write\_file, NULL);

pthread\_create(files, NULL, aggregate\_files, NULL);

pthread\_exit(NULL);

}

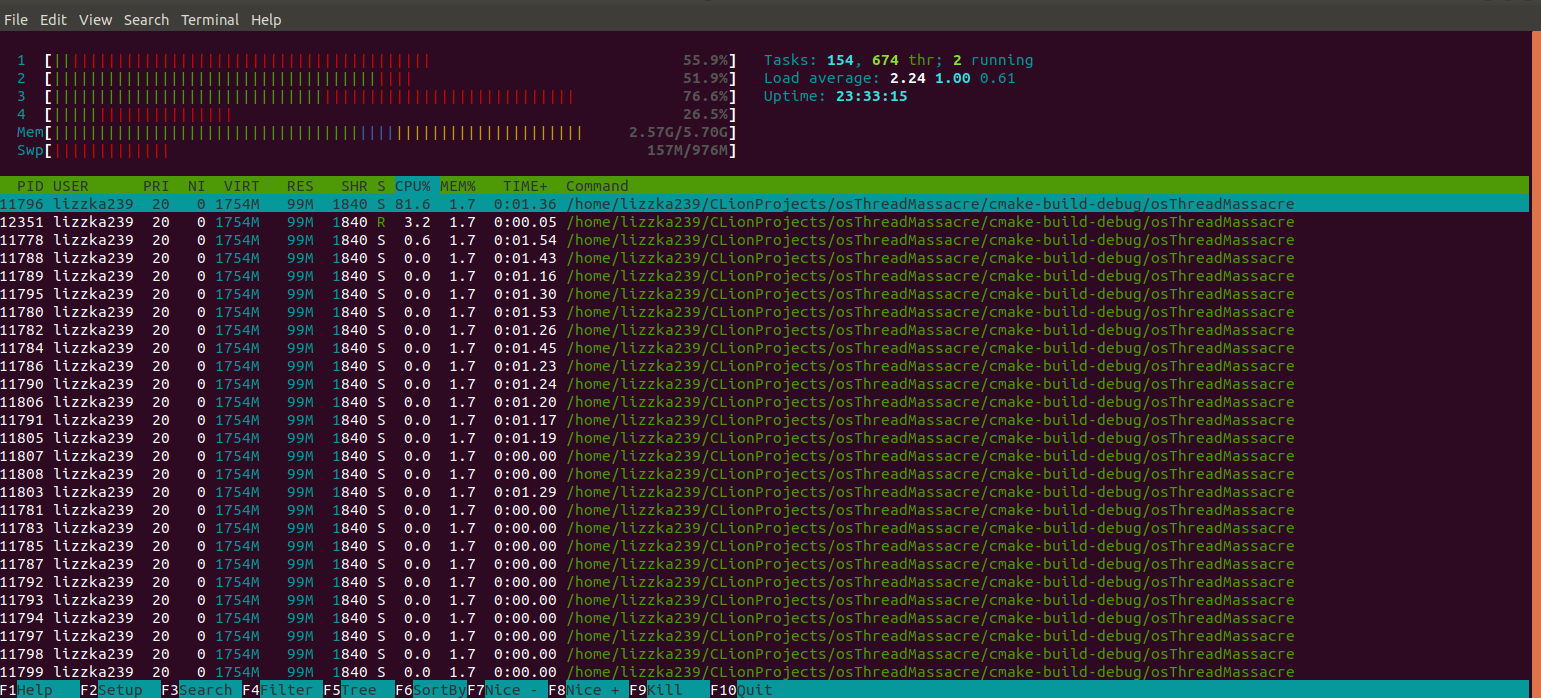
**Ход выполнения.**

Мониторинг и профилирование выполнялись в Ubuntu 18.04. Средствами мониторинга являлись утилиты gdb, htop (top не хотел показывать ресурсы потоков).

**Результаты замеров потребляемой памяти.**

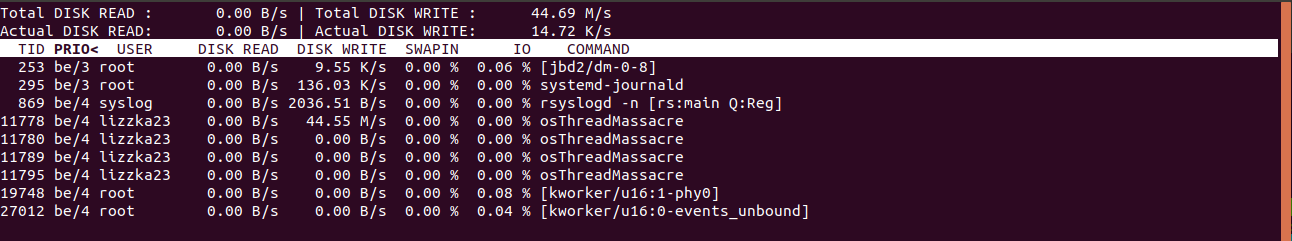
|  |  |
| --- | --- |
| **Точка замера** | **Результат замера** |
| До аллокации |  |
| После аллокации |  |
| После заполнения данными |  |
| После деаллокации |  |

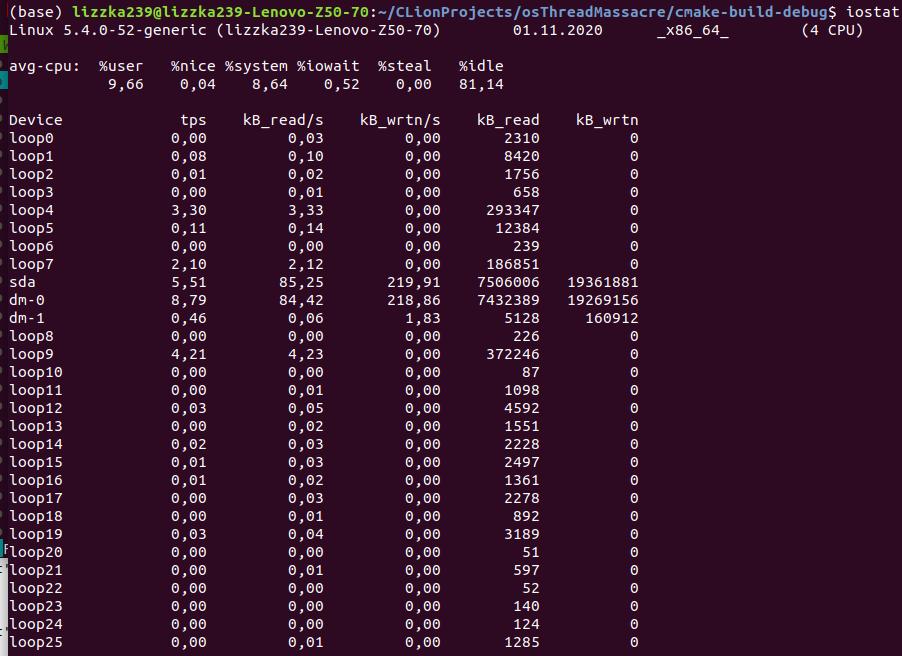
Загрузка CPU:



Статистика IO:

Среднее за 10с (при замере 1 раз в секунду загрузка доходила до 100МБ/с):





**Вывод:**

В результате выполнения лабораторной работы я понял, как можно осуществлять профилирование и мониторинг приложений в OS \*nix. Изучил основы многопоточного программирования на языке С.